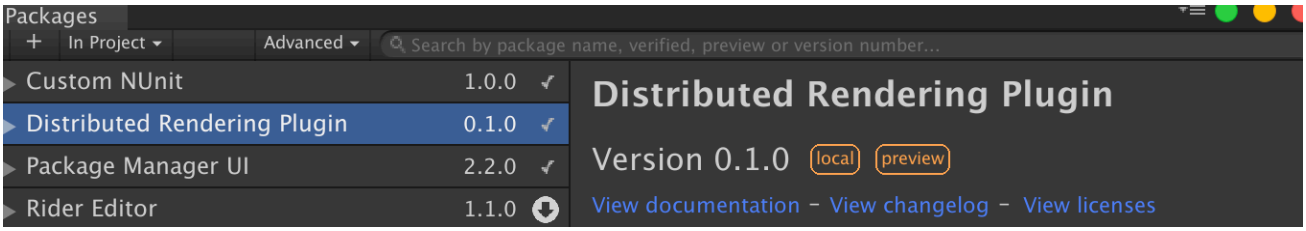
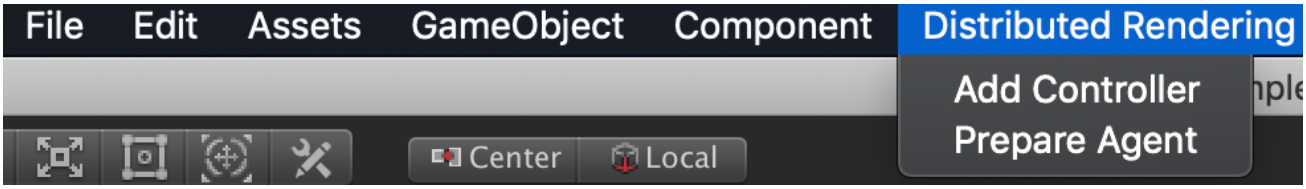
Distributed Rendering & UDB Process

1. 在 Unity 工程中加入 Distributed Rendering 相关插件。
   下图所示为通过 Package Manager 导入已存在于本地的插件，点击 Packages 下方标记找到文件。

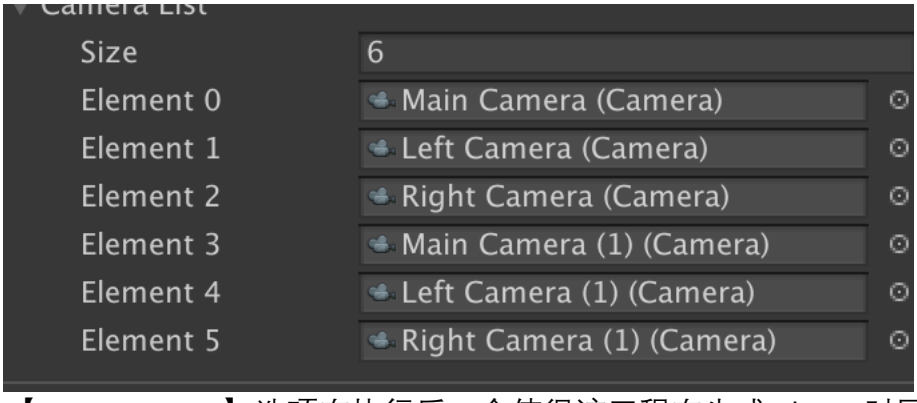
   添加后如图。



2. 加入插件后产生新的菜单项 Distributed Rendering。然后执行对应的两个选项以准备分布式的运行。



   【Add Controller】选项可以在当前场景添加一个带有控制器的 GameObject，通过 Inspector 可以设定其相机数量以及对应编号的相机。该 GameObject 只能生成一个。



   【Prepare Agent】选项在执行后，会使得该工程在生成 Player 时同时将 Agent（即接收任务的程序）打包在同一目录。

3. 生成 Player，然后确认或修改好目录下的 agent.json 中的网络地址。通过该文件设定分布式运行时的控制节点的地址。

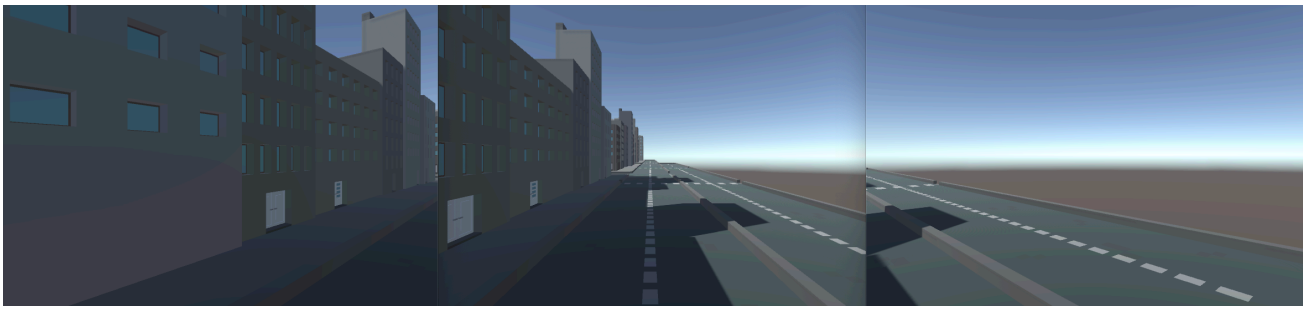
```
{
  "conductorServerHost": "127.0.0.1",
  "conductorServerPort": 8666,
  "agentGroupName": "*"
}
```

然后将 Player 复制至需要运行的机器上，放在路径名相同的位置。然后运行各个 agent。

4. 使用启动器的相关组件。将 client.json 与 conductor.json 中的网络地址设置好。
5. 设置启动器中的 config.json，设置其中的属性：
   - 任务组名称 GroupName
   - Player 的地址与目录
   - 测试阶段 Player 的数量（不含服务器）ClientQty，它与需要的 Camera 数量相等
   - 测试时间长度（秒）TestTime
   - 正式运行时 Player 的数量，它将用于在分析负载后重新分配 Camera
   - 正式运行时需要在相同 Player 一并绘制的 Camera 列表
     以";"隔开每个 Player 的 Camera 列表，相同 Player 的 Camera 之间以":"隔开
     如以"0:1:2:4;3:5"表示有两个 Client 分别绘制 0,1,2,4 号以及 3,5 号两组 Camera
   - 正式运行时间长度（秒）RenderingTime
   - 端口和 Server 的 IP 地址等

```
{
  "GroupName": "0",
  "PlayerPath": "C:\\
  "PlayerFolder": "C:
  "ClientQty": "1",
  "TestTime": "10",
  "RemainedClientQty"
  "EnableList": "",
  "RenderingTime": "6
  "RenderingPort": "4
  "UDBPort": "11001",
  "ServerIP": "10.86.
}
```

6. 运行启动器。
   首先启动 conductor.exe，使得各个它与 agent 连通。
   然后启动触发器。
   系统将先运行一段时间（即 5. 中所述测试时间）动态分析计算负载，然后自动分配节点到不同的计算机上（即 5. 中所述正式运行时间）。下图所示为 3 个 Player 以不同相机角度绘制的同一场景。



7. 运行时，控制分布式的部分可以通过 monitor 监测情况。下图所示为样例界面。